LINKSFOUNDATION.COM

FONDAZIONE links
PASSION FOR INNOVATION

Applied Data Science Project

L6 – Retrieval augumented generation

Politecnico di Torino

ellis
European Laboratory for Learning and Intelligent Systems

Retrieval Augumented Generation (RAG)

RAG is an approach developed in natural language processing for combines two distinct processes: retrieval of external information and text generation

RAG was invented to enhance the performance of generative models (like GPT or T5) by retrieving relevant, real-world information and using it to generate more accurate and contextually relevant responses

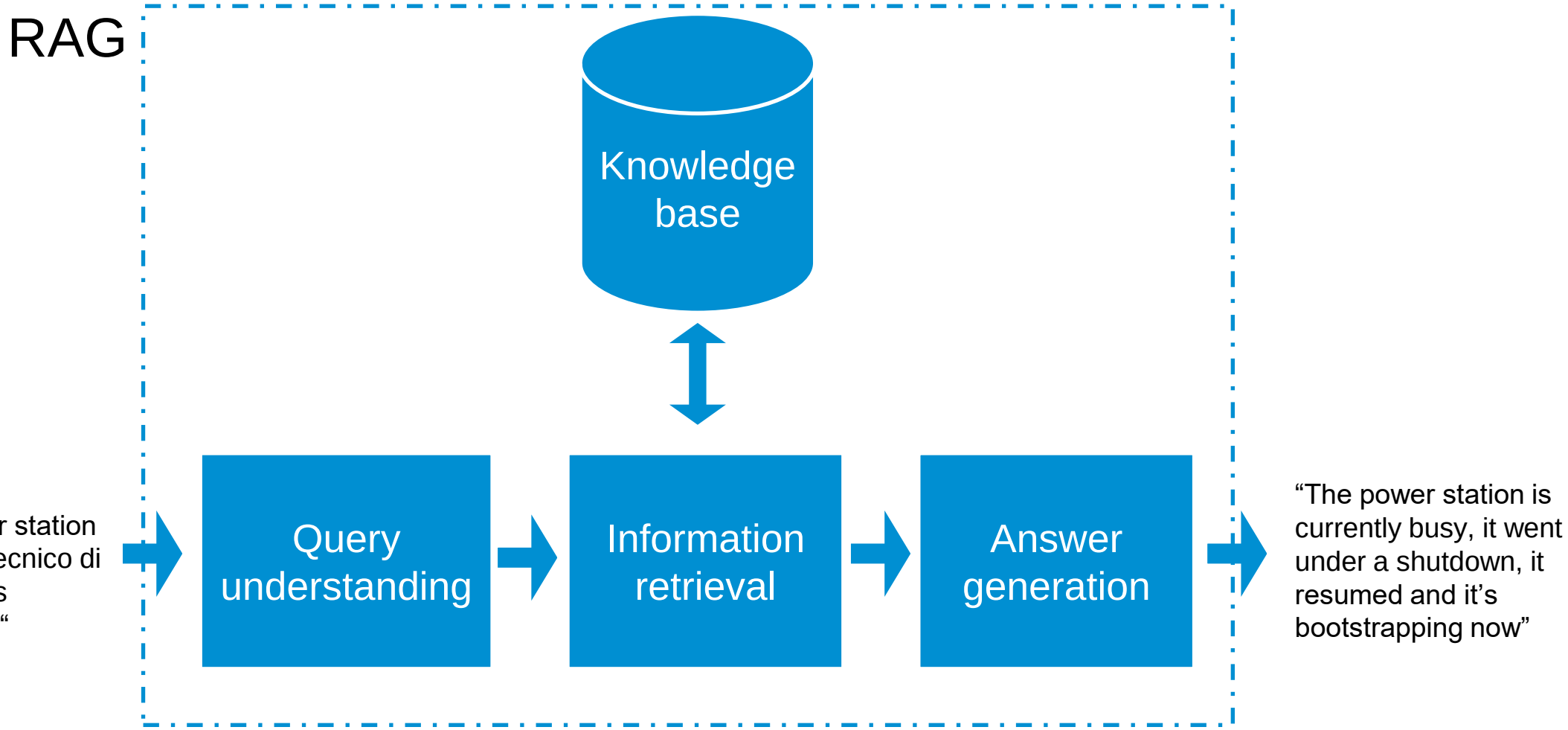
Why?

Traditional generative models rely solely on the data they were trained on, which often has limitations, including the inability to access recent or domain-specific information

Existing problems related to pure generative models:

- **Hallucinations**: the model may generate factually incorrect or plausible-sounding but inaccurate information
- **Limited Knowledge**: the model's understanding is confined to the data present during training, making it less effective for time-evolving or specialized tasks

Functional diagram

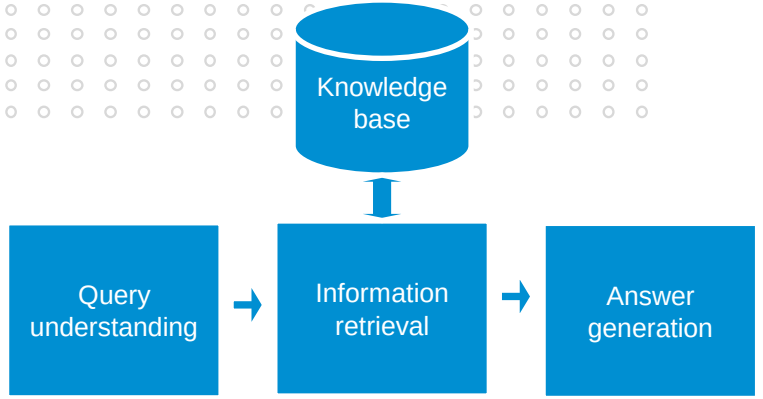


IN/OUT of RAGs

Input: the user submits a query or request, such as a question or prompt

Output: the user receives an answer in natural language

RAG innerworkings



Query understanding: maps the question to a numerical representation

Information retrieval: searches external identified sources, such as a predefined knowledge base, to find relevant information. This could involve traditional search engines, internal databases, or domain-specific documents

Answer generation: instead of generating a response purely from its internal knowledge, the model uses this external data to craft a more accurate, contextually appropriate answer or response

Knowledge base: any information archive in a pre-defined (tabular, or textual) data structure that contains specific knowledge on specific topics that is not mapped in the common knowledge of a large language model

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Patrick Lewis[†‡], **Ethan Perez**[⋆],

Aleksandra Piktus[†], **Fabio Petroni**[†], **Vladimir Karpukhin**[†], **Naman Goyal**[†], **Heinrich Küttler**[†],

Mike Lewis[†], **Wen-tau Yih**[†], **Tim Rocktäschel**[†‡], **Sebastian Riedel**[†‡], **Douwe Kiela**[†]

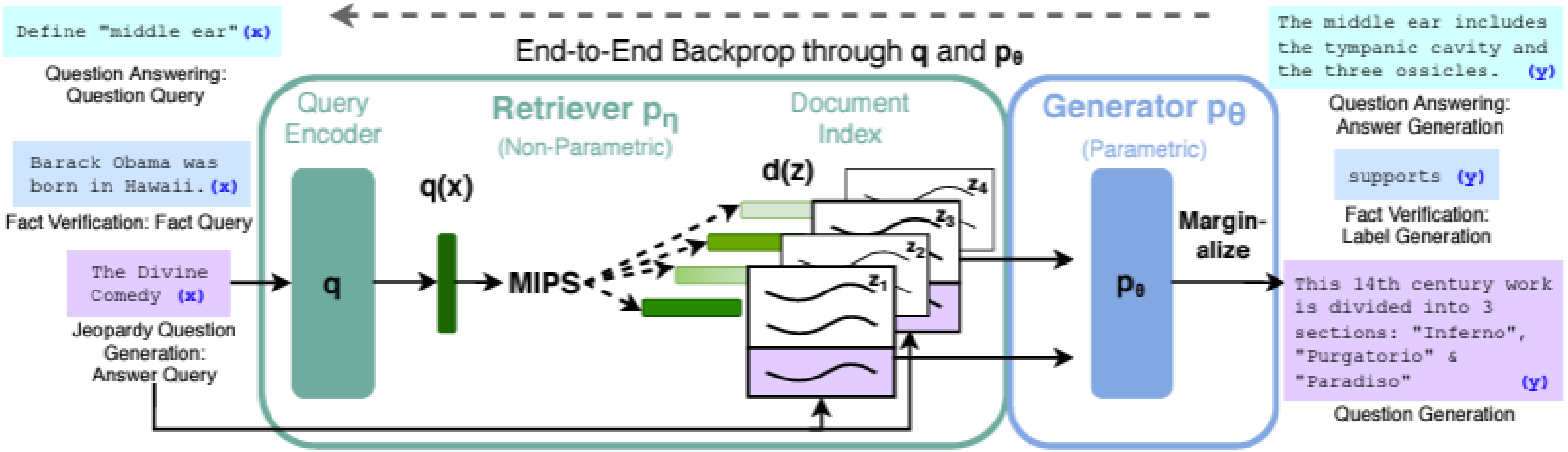
[†]Facebook AI Research; [‡]University College London; [⋆]New York University;
plewis@fb.com

Abstract

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric memory for language generation. We introduce RAG models where the parametric memory is a pre-trained seq2seq model and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We compare two RAG formulations, one which conditions on the same retrieved passages across the whole generated sequence, and another which can use different passages per token. We fine-tune and evaluate our models on a wide range of knowledge-intensive NLP tasks and set the state of the art on three open domain QA tasks, outperforming parametric seq2seq models and task-specific retrieve-and-extract architectures. For language generation tasks, we find that RAG models generate

https://arxiv.org/abs/2005.11401

Encoder, retriever, database, and generator



https://arxiv.org/abs/2005.11401

Key technologies: retriever (I)

Responsible for finding relevant information from external sources, such as documents, databases, or knowledge bases, to support the generation process.

Implementations:

- Sparse retrieval methods
- Dense retrieval methods

Key technologies: retriever (II)

Sparse retrieval methods, like BM25 (Best Matching 25)

- Are traditional term-matching algorithms
- Usually implemented as a bag-of-words model where each document is represented as a set of keywords, and the search query retrieves documents based on exact or partial keyword matches.
- BM25 ranks documents based on keyword frequency and relevance, focusing on how often certain terms appear and their importance in the document

Pros: It is simple, interpretable, and works well when documents and queries have significant term overlap

Cons: Sparse methods often miss relevant documents if the query doesn't exactly match the terms used in the document (vocabulary mismatch problem). They rely heavily on keyword matching, which limits their ability to retrieve semantically similar content

Key technologies: retriever (II)

Dense Retrieval (e.g., Dense Passage Retrieval):

- It uses neural networks to map both queries and documents into dense vector representations (embeddings)
- usually it captures semantic similarities between the query and documents. A query and the documents are encoded as high-dimensional vectors, and the system retrieves documents whose vectors are close to the query vector in the semantic space (using methods like cosine similarity)

Pros: Dense retrieval is better at capturing semantic relationships, making it more effective in finding relevant content even when there is no direct keyword overlap
Cons: It requires more computational resources (due to the neural network-based embeddings) and training data, and its results are less interpretable than sparse methods

Key technologies: encoder and generator (I)

Both GPT (Generative Pre-trained Transformer) and T5 (Text-To-Text Transfer Transformer) are responsible for producing human-like text. These models are trained on large datasets and learn to predict the next word or phrase given a prompt

How They Work in RAG: In RAG, the generative model does not rely solely on its internal knowledge. Instead, it takes the retrieved information from the retrieval mechanism as input and uses it to generate a more informed, accurate, and relevant response

For Example: if the task is to answer a question about recent developments in AI, the retrieval mechanism fetches relevant documents or snippets from external sources (e.g., recent research papers or news articles). The generative model then reads this information and generates a coherent response based on both the query and the retrieved documents

Key technologies: encoder and generator (II)

Benefits of Using generative models in RAG:

- Contextual Understanding: models like GPT and T5 excel at understanding the context of the query and combining it with the retrieved information to generate a more relevant response

- Fluency: these models produce fluent, natural-sounding language, which makes the output easy to read and understand

- Adaptability: by integrating with retrieval systems, generative models can adapt to new information without needing to be retrained from scratch

Key technologies: database

The database may assume different formats

For instance, in “Retrieval-Augmented Generation for Knowledge-Intensive NLP tasks” the database is defined as a collection of textual documents

Anyway, it can be any dataset built with different technologies storing multimodal data such as:
- Relational database
- Document storage
- Knowledge graph

It is anyway recommended that the storage con cope with the saving of vectors and linked documents of any type, in other words implementing a storage that follows the structure of a dictionary in python

Wrap up on benefits of RAGs

Enhanced Accuracy:

- RAG can be seen as a generative model informed by real-time or up-to-date data

Scalability:

- RAG can be applicable across various domains, like customer support, search engines, and academic research, it’s a matter of loading effectively the database and pick an effective generator

Reduction of Hallucination:

- RAG minimizes incorrect information often generated by purely generative models

Challenges in RAG

Quality of retrieval:

- There are intrinsic limitations if retrieval fails or retrieves irrelevant information

Latency:

- Compared to generation, RAG needs of adding a further step of retrieval, thus it is crucial to balancing speed between retrieval and generation

Complexity:

- There is a complexity in managing multiple systems (encoder, retriever, generator, knowledge base) altogether and effectively

Applications of RAG (I)

Question Answering Systems

Use Case: customer support

RAG application: RAG systems can retrieve relevant documents from a knowledge base or the web, then generate accurate, contextually appropriate answers to user queries. This is particularly valuable for answering complex or fact-based questions that require real-time data or information beyond the model's training cut-off

Example: A customer service chatbot retrieving recent product updates and using that data to generate a personalized response to a customer query

Applications of RAG (II)

Legal Document Processing

Use Case: Legal research and contract analysis

RAG application: legal professionals can use RAG to retrieve relevant case law, statutes, or contract clauses and then generate legal opinions or summaries. This reduces the time spent manually searching for precedents and interpreting complex legal texts

Example: a lawyer uses a RAG system to retrieve relevant cases and generate a legal brief summarizing how past judgments relate to a current case

Applications of RAG (III)

Software coding

Use Case: software development

RAG application: RAG systems can help developers by retrieving relevant code snippets, technical documentation, or tutorials and generating custom code or explanations based on user queries. This improves the efficiency of software development and technical writing

Example: a developer inputs a problem into a RAG system, which retrieves relevant code samples from repositories like GitHub and generates a custom solution based on the developer's specific requirements

Applications of RAG (IV)

Healthcare and Medical Research

Use Case: medical literature review

RAG application: RAG systems can retrieve the latest medical research or clinical guidelines and generate suggestions for diagnosis, treatment options, or patient care. Doctors can input patient symptoms, and the system retrieves similar cases and generates informed recommendations

Example: a clinician inputting symptoms into a RAG system that retrieves relevant journal articles and provides a synthesized diagnosis or treatment plan based on recent findings

FONDAZIONE links

PASSION FOR INNOVATION

Thank you for your attention.

Questions?

CONTACTS

Giuseppe Rizzo

Program Manager (LINKS Foundation) and
Adjunct Professor (Politecnico di Torino)

giuseppe.rizzo@polito.it

FONDAZIONE LINKS
Via Pier Carlo Boggio 61 | 10138 Torino
P. +39 011 22 76 150
LINKSFOUNDATION.COM